The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications

John Knight,[†] Dennis Heimbigner,[‡] Alexander L. Wolf,[‡]
Antonio Carzaniga,[‡] Jonathan Hill,[†] Premkumar Devanbu,[*] and Michael Gertz[*]

[†]Dept. of Computer Science
University of Virginia
Charlottesville, VA 22904-4740 USA
{knight,jch8f}@cs.virginia.edu

[‡]Dept. of Computer Science
University of Colorado
Boulder, CO 80309-0430 USA
{dennis,carzanig,alw}@cs.colorado.edu

[*]Dept. of Computer Science
University of California
Davis, CA 95616-8562 USA
{gertz,devanbu}@cs.ucdavis.edu

University of Colorado
Department of Computer Science
Technical Report CU-CS-926-01 December 2001

ABSTRACT

The Willow architecture is a comprehensive approach to survivability in critical distributed applications. Survivability is achieved in a deployed system using a unique combination of (a) fault avoidance by disabling vulnerable network elements intentionally when a threat is detected or predicted, (b) fault elimination by replacing system software elements when faults are discovered, and (c) fault tolerance by reconfiguring the system if non-maskable damage occurs. The key to the architecture is a powerful reconfiguration mechanism that is combined with a general control structure in which network state is sensed, analyzed, and required changes effected. The architecture can be used to deploy software functionality enhancements as well as survivability. Novel aspects include: node configuration control mechanisms; a workflow system for resolving conflicting configurations; communications based on wide-area event notification; tolerance for wide-area, hierarchic and sequential faults; and secure, scalable and delegatable trust models.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **DEC 2001**	2. REPORT TYPE	3. DATES COVERED **-**

4. TITLE AND SUBTITLE **The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications**	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Defense Advanced Research Projects Agency,3701 North Fairfax Drive,Arlington,VA,22203-1714**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S) 11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Willow architecture is a comprehensive approach to survivability in critical distributed applications. Survivability is achieved in a deployed system using a unique combination of (a) fault avoidance by disabling vulnerable network elements intentionally when a threat is detected or predicted, (b) fault elimination by replacing system software elements when faults are discovered, and (c) fault tolerance by reconfiguring the system if non-maskable damage occurs. The key to the architecture is a powerful reconfiguration mechanism that is combined with a general control structure in which network state is sensed, analyzed, and required changes effected. The architecture can be used to deploy software functionality enhancements as well as survivability. Novel aspects include: node configuration control mechanisms; a workflow system for resolving conflicting configurations; communications based on wide-area event notification; tolerance for wide-area, hierarchic and sequential faults; and secure, scalable and delegatable trust models.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**22**	

1. Introduction

As a society, we are becoming increasingly dependent on the continuous, proper functioning of large-scale, heterogeneous, distributed information systems. These systems are formed from large numbers of components, both hardware and software, originating from multiple sources assembled into complex and evolving structures spread across wide geographic areas. Our goal is to develop techniques that help to either prevent disruptions or ensure that these systems can continue to provide acceptable though not necessarily complete levels of service, that is to *survive*, in the face of serious disruptions to their normal operation. In this paper we present the main features of the *Willow architecture*, an architecture that provides a comprehensive architectural approach to survivability [14] in critical distributed applications. The important contribution of the architecture is the merging of three major approaches to dealing with faults into a single system. Although this paper only summarizes the architecture, other details are available in other cited papers.

The Willow architecture is based on the notion that survivability of a distributed application requires a broad approach to dealing with faults in the application, an approach that includes fault avoidance, fault elimination, and fault tolerance. Thus it includes mechanisms: (a) to *avoid* the introduction of faults into the systems at the time of initial deployment or subsequent enhancement; (b) to *eliminate* (i.e., remove) faults from a deployed application once they are either identified or merely suspected but before they can cause failure; and (c) to *tolerate* the effects of faults during operation. These various mechanisms are all based on a general notion of *reconfiguration* at both the system and the application levels together with a framework that implements a monitor/analyze/respond approach to the identification and treatment of faults.

Experience has shown that serious faults are often introduced during system deployment and enhancement. For example, software components are distributed with default passwords set, the wrong software version is deployed, corrective patches are not applied, and so on. Prevention and repair of such faults are the reason that fault avoidance and fault elimination are needed. A novel special case of fault elimination occurs in circumstances where a fault is suspected but not diagnosed. In such cases, it is desirable for the component(s) with the fault to be removed from the system if circumstances indicate that the fault might be manifested. As an example, consider worm attacks against servers that have been vulnerable to worms in the past. If there is a possibility that some servers might still be vulnerable and a new worm is detected, rapidly disconnecting all the servers in the suspect class from the network until the worm is eliminated might be a prudent precaution. In this case, a fault is suspected and eliminated temporarily, although during the period that the fault is eliminated, functionality probably had to be changed. This particular form of fault elimination is sometimes referred to as *posturing*.

Although fault avoidance, fault elimination, and fault tolerance are related but different concepts, each is an approach to dealing with faults, and all three are provided by Willow. Their implementations share a number of architectural elements and a common architectural theme of reconfiguration. Despite all three being designed to deal with faults, they are not entirely compatible and actions taken by one can conflict with actions taken by another. For example, if a fault such as a widespread power failure occurs while an activity is underway to eliminate a fault by replacing software components, it might be necessary to reconfigure the system to deal with the power loss as a higher priority than the ongoing software replacement. The Willow architecture contains mechanisms to deal with such conflicts.

Security of the Willow architecture and its data sources is of paramount importance because, if it were compromised, an intruder could do immense damage. Various techniques have been developed to support the Willow architecture and are described.

This paper is organized as follows. In section 2, the characteristics of the applications of interest are discussed and the notion of survivability reviewed. The special characteristics of the faults with which we are concerned are summarized in section 3, and in section 4 we present the concepts of the Willow architecture. A detailed discussion of the architecture is presented in section 5 and an evaluation in section 6. Finally, we review related work briefly in section 7, and present our conclusions in section 8.

2. Critical distributed applications and survivability

The distributed information systems with which we are concerned arise in the context of the nation’s critical infrastructures. Transportation, telecommunications, power distribution and financial services are examples, and such services have become essential to the normal activities of society. Similarly, systems such as the Defense Department’s Global Command and Control System (GCCS) are essential to the nation’s defense operations.

In such a system, all or most of the service it provides can be lost quickly if certain faults arise in its information system. Since societal dependence on critical infrastructures is considerable, substantial concern about the dependability of the underlying information systems has been expressed, particularly their security [17, 19].

Critical information systems are typically networks with very large numbers of heterogeneous nodes that are distributed over wide geographic areas [13]. It is usually the case that these systems employ commodity hardware, and that they are based on COTS and legacy software. The networks are often private, and implement point-to-point connectivity (unlike the full connectivity of the Internet) because that is all that is required by the associated applications. Within this structure, there are different numbers of the different types of node, and the different types of node provide different functionality. Some nodes provide functionality that is far more critical than others leading to a situation where the vulnerabilities of the information system tend to be associated with the more critical nodes.

An important architectural characteristic of many critical information systems is that provision of service to the end user frequently involves several nodes operating in sequence with each supplying only part of the overall functionality. This *composition of function* can be seen easily in the processing of checks in the nation’s financial payment system where the transfer of funds from one account to another involves user-specific actions at local banks, transaction routing by regional banks, bulk funds transfer by central banks, and complex signaling and coordination that assures fiscal integrity [21].

The Willow architecture is designed to enhance the survivability of critical information systems. Information-system survivability has been defined in detail elsewhere [14], and so we merely review the meaning of the term here. Informally, the concept is: (1) an information system should provide its complete functionality for some, possibly pre-specified, fraction of the time; (2) the system should meet pre-defined reduced or *different* requirements if it cannot provide full functionality because of failures (including security attacks); and (3) several sets of different requirements might be stated to permit useful if limited functionality to be specified to accommodate different degrees of damage. Given that failures are inevitable, it is essential that attention be paid to how failures are handled and, in particular, how the systems' users will be served during these times. Note that survivability is not a different name for the existing term “graceful degradation”. The essential distinctions are: (1) the possibility of different rather than merely reduced functionality; and (2) the precise statement of these different functionalities in user-defined requirements.

3. Types of fault

3.1 Non-local faults

Traditional fault avoidance and fault elimination techniques such as programming languages with strong type checking and systematic inspection can be used in the development of individual components for critical information systems. Similarly, to effect component- or node-level fault tolerance designers can employ traditional techniques such as N-modular redundancy in processing, communications, data storage, power supplies, and so on. These techniques cope well with a wide variety of faults but in this research we are not concerned with faults that affect a single hardware or software component or even a complete application node. We refer to such faults as *local*, and, although they are important, we assume that local faults are dealt with to the extent possible by existing techniques.

We are concerned with the need to deal with faults that affect significant fractions of a network, faults that we refer to as *non-local*. Some examples of non-local faults are: (1) several nodes having software components lacking essential security patches, defective virus databases, or improperly configured operating systems: (2) extensive loss of hardware or a widespread power failure; (3) failure of an application program; (4) coordinated security attacks, and so forth. Non-local faults are much more difficult to deal with than local faults for obvious reasons.

3.2 Complex faults

In critical information systems, non-local faults are only a small part of the problem. The scale of modern critical applications raises the following three additional issues:

- *Fault Sequences*

  It is necessary to deal with fault *sequences*, i.e., new faults arising while earlier faults are being dealt with. The reason that this is a serious complication is that responses to faults in some cases will have to be determined by the overall application state at the time that the fault is manifested.

- *Fault Hierarchies*

  Any non-local fault must be dealt with appropriately. However, once the effects of a fault are detected, the situation might deteriorate or more information might be obtained leading to the subsequent diagnosis of a more serious fault requiring more extensive action. This suggests the notion of a fault hierarchy and any approach to fault elimination or fault tolerance must take this into account.

- *Interdependent Application Faults*

  Many critical infrastructure applications depend upon one another. For example, the nation's financial system depends upon commercial electric sources and upon commercial telecommunications facilities. Faults and the resulting losses of service in the electrical network could cause losses of financial services if electric supply were interrupted for protracted periods even though nothing was wrong with the financial system itself.

4. Willow concepts

Our approach to dealing with complex, non-local faults is to use an extended form of a particular system architecture that is referred to as a *survivability architecture* or an *information survivability control system* [20]. A survivability architecture is characterised by having as its basic structure a control loop that operates during system execution to monitor the system, analyze it, and effect some form of response if analysis reveals a problem. This monitor/analyze/respond structure is the basis of many network intrusion detection systems, for example [18].

The Willow architecture generalizes the control concept and includes multiple inter-operating but separate loops. In addition, during the execution of the critical application, the state of its *operating environment* is monitored along with the state of the application itself, and the resulting integrated set of state information is analyzed. Necessary changes to the critical application system are effected as required.

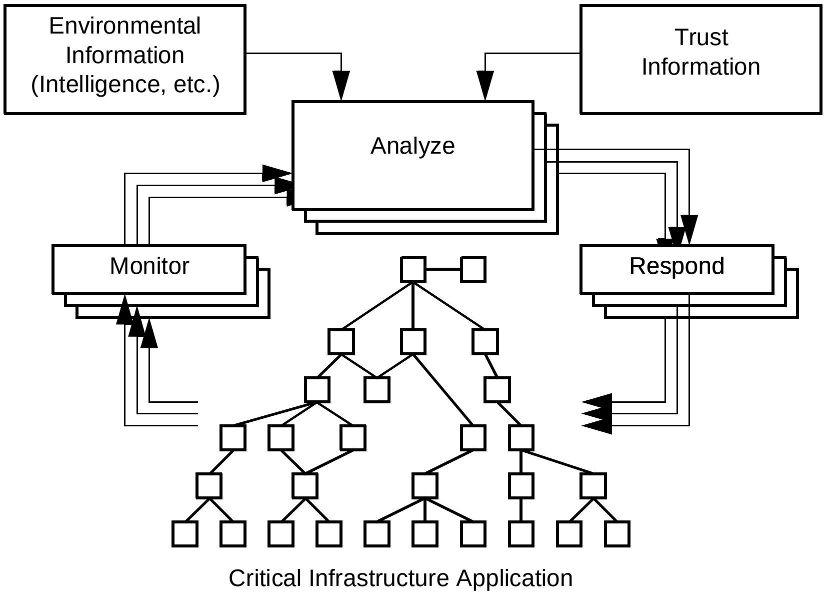
Environmental Information (Intelligence, etc.)
Trust Information
Analyze
Monitor
Respond
Critical Infrastructure Application

Figure 1 Willow survivability architecture concept

In the case of the Willow architecture, all of these notions are applied in a very general sense. Application system state, for example, includes the detailed software configurations in use on the various nodes. The state of the operating environment includes intelligence information about security threats together with details about new releases of the software components used within the system. Finally, necessary changes include correcting defective operating system or application configurations.

Changes to the critical application system include both software updates that will be transparent to the system's user and modifications to the system's functionality that will not be transparent. Functionality changes include reducing some services, ceasing others, and perhaps initiating application services that are not normally available (such as basic emergency services). Monitoring and change are carried out by sensing and actuating software that resides on network elements of interest. Analysis is performed by servers that are not part of the application network, and communication between the monitored nodes and the servers is by independent communications channels. The Willow architecture concept including multiple instances of the basic control loop is illustrated in Figure 1.

The necessary changes in a critical application are effected by *reconfiguration.* The Willow architecture supports reconfiguration in a very broad sense, and reconfiguration in this context refers to any scenario that is outside of ongoing, “steady-state” operation. Thus, for example, initial system deployment is included intentionally in this definition as are system enhancements and upgrades, posturing, recovery from hardware failure, and so on. All are viewed merely as special cases of the general notion of reconfiguration. We refer to reconfiguration that is effected before the system has sustained any damage as *proactive* and after the system has sustained damage as *reactive*. Proactive reconfiguration adds, removes, and replaces components and interconnections, or changes their mode of operation. In a complementary fashion, reactive reconfiguration adds, removes, and replaces components and interconnections to restore the integrity of a system in bounded time once damage or intrusions have taken place. Examples of specific system reconfigurations that can be supported by Willow are:

- Application and operating system updates including component replacement and re-parameterization. Initial application system deployment is treated as a special case of this.
- Planned posture changes in response to anticipated threats.
- Planned fault tolerance in response to anticipated component failures.
- Systematic best efforts to deal with unanticipated failures.

Since reconfiguration could be used as a means of security attack, the input that is used in the Willow decision-making process is managed by a comprehensive trust mechanism [5]. In addition, extensive protection is used for the elements of the Willow architecture itself.

The Willow concept derives from a realization that system and application configuration control and application fault tolerance are two different aspects of the general problem of overall control of distributed systems. Both utilize specialized knowledge about the applications, the resources available and the application state to prepare and react to changing conditions for distributed applications. The difference lies in the time frames at which the two aspects operate, and in the mechanisms used to detect and respond to circumstances needing action. Application fault tolerance is mostly time-bounded, needing prescribed responses to anticipated faults. Software configuration management involves run-time analysis of application state to determine necessary basic actions from a series of prescribed facts and newly available application elements (new software versions, operating system conditions, etc.)

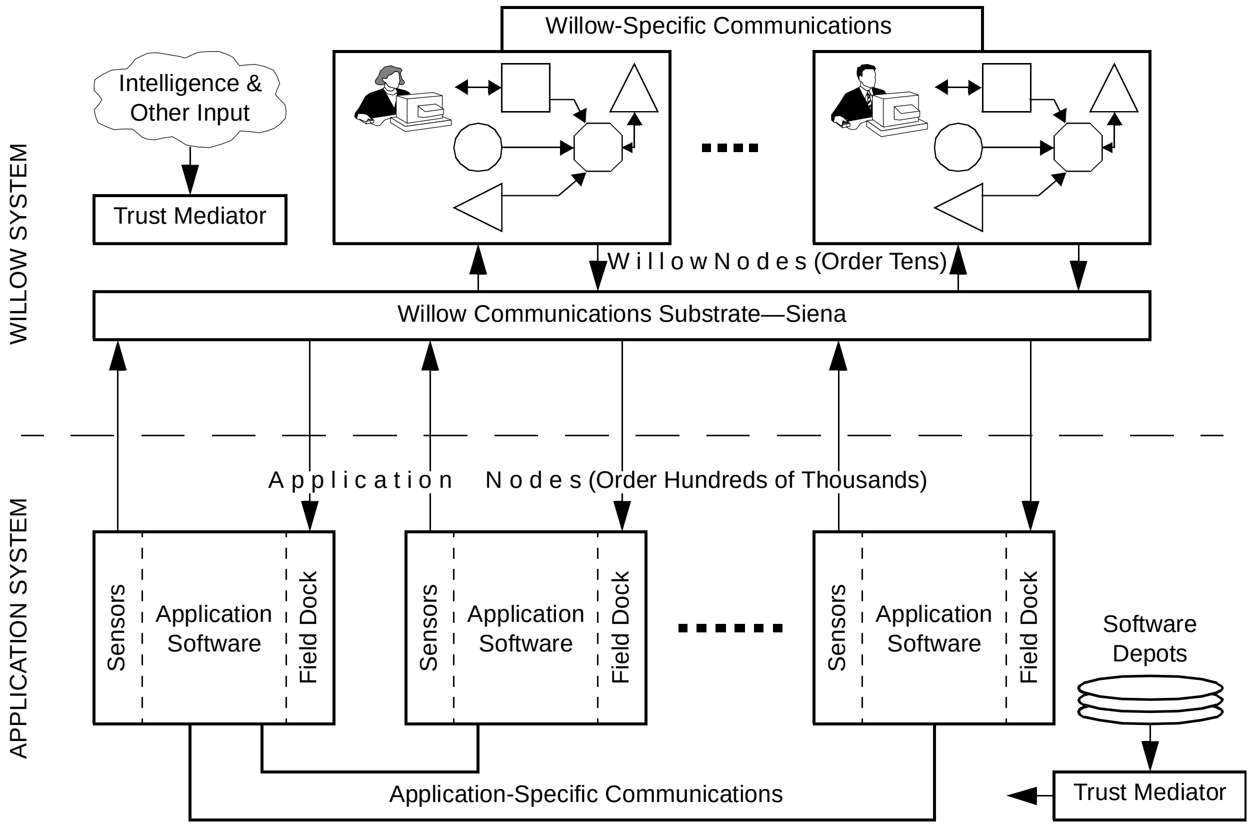
Willow-Specific Communications
Intelligence & Other Input
Trust Mediator
WILLOW SYSTEM
W i l l o w N o d e s (Order Tens)
Willow Communications Substrate—Siena
A p p l i c a t i o n N o d e s (Order Hundreds of Thousands)
APPLICATION SYSTEM
Sensors
Application Software
Field Dock
Software Depots
Trust Mediator
Application-Specific Communications
LEGEND
□ Admin Workbench
◁ RAPTOR Analysis
○ Other Analysis
⬡ Workflow Manager
△ Resource Manager

Figure 2 The Willow architecture showing major components and execution-time event communications

5. The Willow architecture

5.1 Control loops

The major components and the major forms of communication in the Willow architecture are illustrated in Figure 2. The top part of the figure shows the majority of the Willow system and the lower part shows the distributed application that Willow is supporting. The distributed application operates on a traditional network and has computation and communication requirements that are met by an unspecified (but assumed large) set of nodes together with appropriate communications links. The Willow system operates (at least in this example) on a separate network, the details of this network will depend upon the computational and communication needs of the specific Willow system for the application.

The fundamental structure of the Willow architecture, the set of *control loops*, has sensing, diagnosis, synthesis, coordination, and actuation components and these are depicted in Figure 2. The control loops begin with a shared *sensing* capability shown within the application nodes. Sensors can include reports from applications, application heartbeat monitors, intrusion detection alarms, or any other means of measuring actual application properties.

From sensing events, independent *diagnosis and synthesis* components build models of application state and determine required application state changes. In the current Willow architecture, there are two of these components—the Administrator's Workbench for proactive reconfiguration and RAPTOR for reactive reconfiguration. Additional diagnosis and synthesis components can be added easily and this is illustrated in the figure by the "*Other*" analysis component.

Synthesis components issue their intended application changes as *workflow* requests. These are coordinated by the workflow and the resource managers to ensure that changes occur correctly and smoothly within the application.

When workflows are allowed to activate, workflow events are received by the Field Docks located within the application nodes and result in local system state changes. The Field Dock infrastructure provides a single standard interface for *universal actuation* at application nodes [8, 9, 10]. Actuation completes the control loop cycle.

5.2 Proactive control

The proactive controller, the Administrative Workbench, is an interactive application allowing system administrators to monitor system conditions remotely, adjust system properties, and most importantly, cause the propagation and implementation of proactive reconfigurations.

The Software Depot in Figure 2 represents an external source of information that may be required by Willow in order to complete its reconfigurations. In general, there will likely be many such depots. This information may include models of new applications, components needed in a new configuration (but which are not already available locally on the affected application node), and components that provide additional actuators to, for example, access built-in reconfiguration capabilities for specific applications.

5.3 Reactive control

The reactive controller, known as RAPTOR, is a fully automatic structure that is organized as a set of *finite state machines.* The detection of the erroneous state associated with a fault (i.e., error detection) is carried out by a state machine because an erroneous state is just an application system state of interest. As the effects of a fault manifest themselves, the state changes. The changes become input to the state machine in the form of events, and the state machine signals an error if it enters a state designated as erroneous. The various states of interest are described using predicates on sets that define part of the overall state. The general form for the specification of an erroneous state, therefore, is a collection of set definitions that identify the application objects of concern and predicates using those sets to describe the various states for which either action needs to be taken or which could lead to states for which action needs to be taken.

In an operating application, events occurring at the level of individual application nodes are recognized by a finite-state machine at what amounts to the lowest level of the system. This is adequate, for example, for a fault like a wide-area power failure. Dealing with such a fault might require no action if the number of affected nodes is below some threshold. Above that threshold might require certain critical application nodes to respond by limiting their activities. As node power failures are reported so a predefined set, say *nodes_without_power*, is modified, and once its cardinality passes the threshold, the recognizer moves to an error state.

The notion of a fault hierarchy requires that more complex fault circumstances be recognized. A set of nodes losing power in the West is one fault, a set losing power in the East is a second, but both occurring in close temporal proximity might have to be defined as a separate, third fault of much more significance because it might indicate a coordinated terrorist attack. This idea is dealt with by a *hierarchy* of finite-state machines. Compound events can be passed up (and down) the hierarchy, so that a collection of local events can be recognized at the regional level as a regional event, regional events could be passed up further to recognize national events, and so on. As an example, a widespread coordinated security attack might be defined to be an attack within some short period of time on several collections of nodes, each within a separate administrative domain. Detection of such a situation requires that individual nodes recognize the circumstances of an attack, groups of nodes collect events from multiple low-level nodes to recognize a wide-area problem, and the variety of wide-area problems along with their simultaneity recognized as a coordinated attack.

5.4 Dealing with conflicting goals

During operation of a distributed application in a Willow system, reconfiguration could be initiated to avoid a fault (posturing), to eliminate a fault (classical reconfiguration in which some set of software component are updated in some way to correct a problem), or to tolerate a fault (physical damage, security attack, etc.). Thus there are multiple sources of initiation for reconfiguration and they are asynchronous. Clearly, this means that more than one might be initiated at the same time in which case either one has to be suspended or there has to be a determination that they do not interfere. Worse, however, is the prospect that one or more new initiations might occur while already initiated reconfigurations are underway. Some reconfigurations are more important than others—tolerating a security attack once detected, for example, is more important than eliminating a minor fault in some piece of application software—and so reconfigurations underway might have to be suspended or even reversed. Unless some sort of comprehensive control is exercised, the system would quickly degenerate into an inconsistent state.

One approach would be to have each source make its own determination of what it should do. The complexity of this approach makes it infeasible. An implementation would have to cope with on-the-fly determination of state and, since initiation is asynchronous, that determination would require resource locking and so on across the network. The approach we have taken is to route all requests for reconfiguration through a resource manager/priority enforcer called ANDREA.

The prototype ANDREA implementation uses predefined prioritization of reconfiguration requests and dynamic resource management to determine an appropriate execution order for reconfiguration requests using a distributed workflow model to represent reconfiguration requests. The workflow model formally represents the intentions of a reconfiguration request, the temporal ordering required in its operation, and its resource usage. Combined with a specified resource model, this information is the input to a distributed scheduling algorithm that produces and then executes a partial order for all reconfiguration tasks in the network.

Scheduling within ANDREA is preemptive allowing incoming tasks to usurp resources from others if necessary so that more important activities can override less important ones. Transactional semantics allow preempted or failed activities to support rollback or failure, depending on the capabilities of the actuators enacting the reconfiguration. ANDREA supports an event-driven interface so that adaptive systems can be easily interface and observe Willow's reconfiguration protocol.

The current ANDREA system demonstrates formal specification of complex reconfiguration tasks across multiple nodes of a distributed application system. Redundant application of complex, yet menial management tasks is completely automated. Multiple administrators and automated controllers can engage in proactive and reactive management of the system without resource conflicts and without the potential of interrupting more important tasks initiated by another controller. A controller that initiates a high priority task is guaranteed to receive resources in bounded time given that actuators for current tasks are compliant with interruptible transaction semantics. In general, the current ANDREA system demonstrates that asynchronous parallel control-loops can interact to enhance, rather than degrade, the survivability of an application system.

5.5 Communication

The communication challenges presented by the Willow architecture are considerable—sensor information has to be transmitted from nodes to the analysis components of the control loops, reconfiguration commands have to be transmitted from the analysis components to the nodes, replacement software components have to be transmitted from Software Depots, and so on. This would not be a significant challenge but for the fact that one of the targets of the Willow architecture is very large networks, i.e., networks with many nodes.

The approach to communication that Willow takes is to reduce as much of the communication as possible to event notification and then to use a highly efficient wide-area event notification service as the communications substrate. The specific event notification service that Willow uses is Siena [4]. Siena uses a broad range of techniques to maximize the efficiency of its communication service including: (1) a high-level router structure to link its various servers; and (2) routing only a single copy of a notification a far as possible, only replicating it when necessary. Siena avoids explicitely a solution based on a central database since such an approach would not scale to the degree that Willow requires and would severely limit applicability.

The major communication service that cannot be reduced to event notification is the transmission of files associated with software or bulk data distribution. This is handled by using events to define and initiated file transfers and then allowing the actual transfers from Software Depots to take place via other network communications services.

5.6 Security

Given the pervasive notion of control in the architecture, there is the risk that an adversary might exploit the Willow infra-structure itself to cripple the distributed application. The defensive measures in the Willow architecture deal with two major concerns. The first is securing the mechanisms of the architecture itself, and the second is ensuring that the information used by the system is current, accurate, and intact. Thus we defend against two types of adversaries—those that seek to directly subvert the control mechanisms, and those who try to do that indirectly by tampering with the data used by the control mechanisms.

Securing the mechanisms of the architecture breaks down into the following three technical problems:

- Protecting the Willow servers that conduct the analysis.
- Protecting the sensors and actuators that reside on application nodes.
- Protecting the communication between the various Willow components.

The first of these three problems can be solved using conventional techniques such as physical security, personnel authentication (via passwords, bio-metrics, etc), and cryptography.

The second problem is much harder to solve. Sensors and actuators operate on a large, heterogeneous collection of nodes located in a variety of administrative domains. We must, therefore, require that trusted components (the sensors and the actuators) operate correctly and continuously on untrustworthy hosts (the application nodes). If the sensor code could be either tampered with or replaced, for example, an adversary could arrange for the transmission of erroneous data for analysis and thereby force malicious control actions.

Protecting the trusted components is achieved using a combination of several approaches. The possibility of an adversary understanding the software by analyzing the binary statically is reduced by code obfuscation. Tamper resistance is further improved by using randomization in the observable behavior of software, and by employing temporal and spatial diversity of the trusted code. By temporal diversity we mean the intentional replacement of the software at periodic interval, with a new version that has different observable behavior. A randomization technique [23] is used to produce the various versions of the software such that only trusted components know the randomization that was used. By spatial diversity we mean the use of intentionally diverse versions of the software whose functionality is nominally identical (the sensors, for example) throughout the system. Production of the diverse versions can be achieved using specially modified compilers that generate the necessary diversity.

The third problem in the list above (securing the communication between the various Willow components) is solved partially by encryption but a new challenge is raised by the Willow architecture—securing the publish/subscribe communication service that underlies the infrastructure [4]. For Willow, the wide-area publish/subscribe service must handle information dissemination across distinct authoritative domains, heterogeneous platforms, and a large, dynamic population of publishers and subscribers. Such an environment raises serious security concerns. The general security needs of the client of a publish/subscribe service include confidentiality, integrity, and availability, while the security concerns of the publish/subscribe infrastructure itself focus primarily on system integrity and availability. A detailed discussion of how some aspects of the problem can be solved is available in a separate publication [23].

The second broad category of security issues (security services to allow the latest, most accurate and most trusted information to be used) is effected by a secure mediator infrastructure (SMI). There are two design goals for the SMI. First, administrators can precisely control and coordinate where and how data are obtained, via a centralized *trust broker* that gathers and disperses meta-data on the trustworthiness of data sources. Second, the SMI allows authentic data distribution [5] without the use of on-line signing keys. Keyless hashing primitives are used to validate the distributed data, with the data signed infrequently using off-line keys. By avoiding on-line keys, which might be stolen by an attacker, we enhance security and reduce the administrative burden. Finally, data mediation allows translation of data from different sources, and selection of data sources based on different criteria over the trust meta-data This SMI design provides scaleability, delegation and separation of concerns, so that the Willow infrastructure can obtain the best information from the widest possible set of sources with good security and low administrative overhead.

6. Evaluation

The Willow architecture is designed to provide a sophisticated service to critical applications operating on large and complex networks. Evaluating the architecture is difficult because of the scales of the parameters involved. Typically, the numbers of nodes in a network will be large, there might be several node types, the functions provided by the application will be sophisticated and there will be many of them, the number of users of the system will be large, and data maintained in databases will generally be extensive.

Ideally, experimental evaluation of the Willow architecture would be conducted on a full-scale implementation that was supporting representative applications with the system characteristics just noted. Clearly this is infeasible and another approach has to be taken. Our approach to evaluation is in two parts. In the first part, the major elements of the architecture have been studied individually using a combination of analysis, small-scale implementation, and simulation. In the second part, a laboratory-scale Willow system has been developed as a case study and to evaluate feasibility. In this section, we summarize briefly the various evaluation efforts of the major elements, and review the laboratory-scale case study.

6.1 Evaluation and status of components

The reconfiguration component of the Willow architecture is derived from the Software Dock system [10]. A prototype Software Dock was implemented as part of a previous research project and its performance evaluated.

The proactive control component—the Administrator’s Workbench—has been developed as a layer on top of the existing Software Dock. It connects to the internal event communication mechanism used by the Software Dock to provide network level control of one or more Field Docks. Since its operation is directly tied to that of the Software Dock, we expect that the Software Dock performance carries over to the Workbench.

The RAPTOR reactive reconfiguration mechanism has been evaluated using two models of critical infrastructure applications. The first model was of the nation’s financial system and the second was of the nation’s electric power grid. In both cases, models were built that implemented abstract versions of the application, including only essential functionality, along with a complete implementation of the error detection and error recovery mechanisms. Both models were of realistic size, each involving thousands of application nodes. These two models were executed using a network simulation system and faults injected. Details of the results can be found elsewhere [6] but the overall conclusion was that the RAPTOR mechanism as modified to operate in the Willow system will provide the desired fault-tolerance capability.

The ANDREA workflow system is the only major component of the Willow architecture that cannot be evaluated easily outside the context of a full-scale implementation. Its performance in real time is the most important aspect of evaluation but this is difficult to measure because it depends on so many elements of a complete implementation. Informal assessment of the performance of the current implementation has been completed and one significant conclusion has been reached—as presently designed, the ANDREA system is not well suited for scaling to large networks because the workflow specifications are not location transparent. The next generation ANDREA system (currently being implemented) will allow scaling of compact workflow specifications to networks of a much larger scale because it will allow run-time, multicast binding of workflow commands to actuation sites rather than requiring explicit location specification.

Many aspects of Siena, our wide-area event notification system, have been studied. A prototype implementation has been built and used in a number of trials by ourselves and by others who have downloaded the implementation. In addition, we have carried out extensive simulation studies to assess how the system performs in terms of scale, throughout, etc. Detailed reports of these assessments have been reported by Carzaniga et al [4]. We conclude from these studies that the anticipated workload imposed upon Siena by the expected operating circumstances of a production Willow implementation are well within its capacity provided the required time bounds on response dictated by the application are within the scope of the system.

6.2 Case study

A prototype Willow system has been developed that implements all the different aspects of the architecture mentioned above except certain aspects of the security mechanism. The system includes an adminstrator's workbench, a RAPTOR error detection mechanism, and a communications infrastructure based on the Siena publish/subscribe system. The application that we have chosen is a very simple prototype implementation of the U.S. Air Force's *Joint Battlespace Infosphere* (JBI) concept [22].

Instantiations of the JBI concept, once fully developed, will provide advanced information systems for military use. At the heart of the JBI concept is the notion of publish/subscribe semantics in which different military information sources publish their data to a network and those interested in the information subscribe to those parts which they wish to see. The expectation is that very large amounst of military information will be published to a JBI and large numbers of consumers (commanders at all levels) will tailor the information they receive by subscribing appropriately. Our JBI implementation is based on Siena and so the feasibilty study implementation uses two entirely separate publish/subscribe systems—one provides the Willow architecture's own essential communication and the second implements the core functionality of our JBI.

A production JBI system will require proper initial deployment, configuration and maintenance of all of its software elements, and will have to be updated quickly and efficiently if defects in the system are observed. In addition, a JBI will be an attractive target to an adversary for many reasons. Such a system might be attacked in various ways by hackers or disabled by battle damage, physical terrorism, software faults, and so on. It is essential, therefore, that a JBI be survivable, and the Willow architecture is a candidate implementation platform.

Our Siena-based JBI implementation includes several information-processing modules (known as fuselets) and synthetic publishers and subscribers. All of the components of the system have been enhanced to allow them to respond appropriately to reconfigration actions. In addition, the different elements of the implementation have been extended deliberately with vulnerabilities so as to permit demonstration and evaluation of the reconfiguration capabilities of Willow.

Operating on the Willow architecture, our JBI implementation has been subjected to a preliminary evaluation by fault injection. The initial deployment of the system to a test network is entirely automatic, and the system has been shown to adopt new postures under operator control as desired. The system has also been shown to reconfigure automatically when network faults were injected.

7. Related Work

All of the component technologies used in in the Willow architecture have been the subject of research although no research has been conducted with the our goal of integrating the components to provide a comprehensive approach to dealing with faults.

Software deployment and configuration management have been studied by numerous researchers. Detailed information on related technologies has been presented by Hall [7]. System-level approaches to fault tolerance are described by several authors [1, 3, 2]. While these efforts are significant in their own right, they do not address non-local faults, and they tend not to address systems as large and complex as critical informations systems.

Reconfiguration has also been studied by others in systems such as CONIC and Darwin [15, 16]. Purtilo and Hofmeister studied the types of reconfiguration possible within applications and the requirements for supporting reconfiguration [11]. Details of related work on event notification services can be found in the paper by Carzaniga et al [4].

8. Conclusion

In this paper we have summarized the major elements of the Willow architecture. The architecture is designed to improve the survivability of critical distributed information systems by providing a range of mechanisms to deal with faults. The unique contribution of the architecture is the combination of fault avoidance, fault elimination and fault tolerance into one integrated system. The benefits of this approach include assurance that: (1) certain types of fault will not be present; (2) certain types will be removed properly once they are found; (3) certain types of faults will be tolerated during execution; and (4) all these activities will be properly coordinated within the system.

In some cases, the way in which faults are handled might require intentional changes in the service supplied to the system’s users but such changes are part of the notion of survivability. The damage that large distributed applications experience or anticipate that they might experience frequently will not permit complete functionality to be maintained. It is essential, therefore, that when dealing with faults provision be made for these different functionalities.

The Willow architecture cannot be evaluated as part of this research in a typical, large-scale, production environment because such environments cannot be studied in a typical laboratory context. Thus, development and analysis of the different components of Willow have been carried out separately with some analysis carried out analytically, some using small-scale implementations, and some using simulation. A laboratory-scale Willow system has been developed as a case study. All of the results todate indicate that the Willow architecture will perform entirely satisfactorily in a production implementation.

9. Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency under grant N66001-00-8945 (SPAWAR) and the Air Force Research Laboratory under grant F30602-01-1-0503. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force, or the U.S. Government.

10. References

[1] Alvisi, L. and K. Marzullo. “WAFT: Support for Fault-Tolerance in Wide-Area Object Oriented Systems,” Proceedings of the 2nd Information Survivability Workshop, IEEE Computer Society Press, Los Alamitos, CA, October 1998, pp. 5-10.

[2] Bhatti, N., M. Hiltunen, R. Schlichting, and W. Chiu. “Coyote: A System for Constructing Fine-Grain Configurable Communication Services,” *ACM Transactions on Computer Systems*, Vol. 16 No. 4, November 1998, pp. 321-366.

[3] Birman, K. “The Process Group Approach to Reliable Distributed Computing,” *Communications of the ACM*, Vol. 36 No. 12, December 1993, pp. 37-53 and 103.

[4] Carzaniga, A., D.S. Rosenblum, and A.L. Wolf, “Design and Evaluation of a Wide-Area Event Notification Service”, *ACM Transactions on Computer Systems*, 19(3):332-383, Aug 2001.

[5] Devanbu, P., M. Gertz, C. Martel, and S. Stubblebine, “Authentic Third-Party Data Publication”, in Proceedings of the Fourteenth IFIP Working Conference on Database Security, August 2000.

[6] Elder, M.C. “Fault Tolerance in Critical Information Systems,” Ph.D. dissertation, Department of Computer Science, University of Virginia, May 2001.

[7] Hall, R.M., “Agent-based Software Configuration and Deployment,” University of Colorado Ph.D. Thesis, April 1, 1999.

[8] Hall, R.M., D.M. Heimbigner, A. van der Hoek, and A.L. Wolf. “An Architecture for Post-Development Configuration Management in a Wide-Area Network” in Proceedings of the 1997 International Conference on

Distributed Computing Systems, pages 269–278. IEEE Computer Soci-ety, May 1997.

[9] Hall, R.M., D.M. Heimbigner, and A.L. Wolf., “Evaluating Software Deployment Languages and Schema”, in Proceedings of the 1998 International Conference on Software Maintenance, pages 177– 185. IEEE Computer Society, November 1998.

[10] Hall, R.M., D.M. Heimbigner, and A.L. Wolf. “A Cooperative Approach to Support Software Deployment Using the Software Dock”, in Proceedings of the 1999 International Conference on Software Engineering, pages 174–183. Association for Computer Machinery, May 1999.

[11] Hofmeister, C., E. White, and J. Purtilo. “Surgeon: A Packager for Dynamically Reconfigurable Distributed Applications,” *Software Engineering Journal*, Vol. 8 No. 2, March 1993, pp. 95-101.

[12] Knight, J.C., M. C. Elder, “Fault Tolerant Distributed Information Systems”, International Symposium on Software Reliability Engineering, Hong Kong (November 2001).

[13] Knight, J., M. Elder, J. Flinn, and P. Marx. “Summaries of Three Critical Infrastructure Systems,” Technical Report CS-97-27, Department of Computer Science, University of Virginia, November 1997.

[14] Knight J.C. and K.J. Sullivan, “On the Definition of Survivability”, University of Virginia, Department of Computer Science, Technical Report CS-TR-33-00.

[15] Magee, J., N. Dulay, S. Eisenbach, and J. Kramer. “Specifying Distributed Software Architectures,” *Lecture Notes in Computer Science*, Vol. 989, September 1995, pp. 137-153.

[16] Magee, J. and J. Kramer. “Darwin: An Architectural Description Language,” http://www-dse.doc.ic.ac.uk/research/darwin/darwin.html, 1998.

[17] Office of the Undersecretary of Defense for Acquisition and Technology. “Report of the Defense Science Board Task Force on Information Warfare - Defense (IW-D),” November 1996.

[18] Porras, P.A. and P. Neumann, “EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances”, Proceedings: 20th National Information Systems Security Conference, October 1997.

[19] President’s Commission on Critical Infrastructure Protection. “Critical Foundations: Protecting America’s Infrastructures The Report of the President’s Commission on Critical Infrastructure Protection,” United States Government Printing Office (GPO), No. 040-000-00699-1, October 1997.

[20] Sullivan, K., J. Knight, X. Du, and S. Geist. “Information Survivability Control Systems,” Proceedings of the 21st International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, May 1999, pp. 184-192.

[21] Summers, B. *The Payment System: Design, Management, and Supervision*, International Monetary Fund, Washington, DC, 1994.

[22] United States Air Force, Joint Battlespace Infosphere Home Page, http://www.rl.af.mil/programs/jbi/default.cfm

[23] Wang, C., A. Carzaniga, D. Evans, and A.L. Wolf , “Security Issues and Requirements for Internet-scale Publish-Subscribe Systems”, in Proceedings of the Thirtyfifth Hawaii International Conference on System Sciences (HICSS-35), Big Island, Hawaii, Jan. 2002.

[24] Wang, C., J. Davidson, J. Hill, J. Knight, “Protection of Software-based Survivability Mechanisms”, International Conference of Dependable Systems and Networks, Goteborg, Sweden (July, 2001)